Tool Support for an Integrated and Multi-Model OS/R Stack*

Martin Schulz (LLNL, schulzm@llnl.gov)

Exascale applications are expected to rely on new programming and execution models and to leverage multiple models concurrently either through some form of hierarchical parallelism, modularization, or MPMD style concurrency. This has a drastic impact on the OS/R stack, which has to support the combination of multiple models for a single application, as well as on programming development tools, which need to provide support to users across all models. Existing conventional approaches (as illustrated in Figure 1 a)) force tools, like in this example a debugger, to interface with each model separately (in addition to the OS) and manually provide the necessary connections between the models, or even worse, lead to multiple independent tools leaving the burden of correlating information entirely to the user. In many cases this is even infeasible at higher levels of abstraction, forcing users to traditional, yet undesirable, `printf` solutions. In order to give users the necessary and required tool support, we need to overcome this ad-hoc model and provide an integrated solution that works in concert with the necessary and expected OS/R integration efforts.

Approach

With the advent of applications that utilize multiple programming or even execution models, users will expect tools that provide easy access to all aspects of their program. In particular, they will expect:

Execution Model Independent Task Control: tools should be able to control any task in any model through a unified abstraction. This should be possible independent of their granularity or their concrete implementation in the OS/R stack.

Programming Model Agnostic Data Abstractions: each programming model provides different abstractions of user data and users will expect to have access to these abstractions across different models. E.g., debuggers should be able to read, write, and monitor data structures in any language; or performance tools should be able to map measurements back to any data source at language level.

Transparent Support for Hybrid Applications: Tool support should not be restricted to single programming models or languages, but any abstraction used by a tool interface should be applicable across models. E.g., watchpoints should be able to depend on information from more than one model and it should be possible to express task dependencies across different execution models.

Correlation with OS/R Events and Profiles: As with programming models, users will also expect to have access to information from the OS/R stack implementing the programming models. Further, users must be able to correlate any data with source level information.

To support these requirements, we need an interface in the OS/R stack that acts as the single entry point for all tools and to which any programming model or OS/R component can register to. This creates a kind of *Tool Bus*, which can act as the central switch and translation point between programming models and tools, as illustrated in Figure 1 b), and that can define and manage the necessary abstractions and mappings.

The challenge in this approach lies in the definition of these abstractions: they need to be abstract enough to be programming and execution model independent to enable a mapping of different task and data abstractions into a common model, yet concrete enough to give tools and consequently users sufficient information. Additionally, any mapping needs to reversible to enable a mapping of information back to user code and the *Tool Bus* interface has to provide the necessary mapping functionality. With the right abstractions in place, though, tools will be able to seamlessly work across an integrated OS/R stack significantly reducing the burden on the user when developing multi-model applications.

*This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

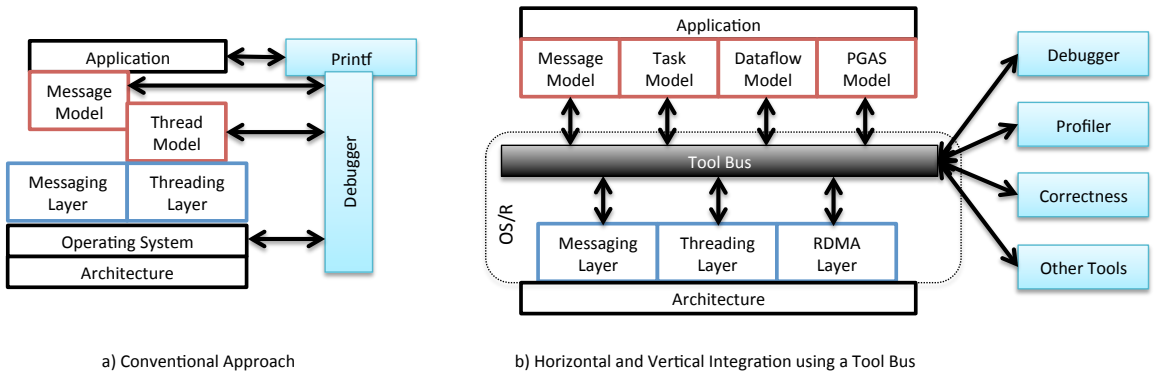

Figure 1: Tool access to the OS/R stack.

Existing Work

Tools support traditionally exists in the OS on a per process basis in the form of debug interfaces (e.g., ptrace on many Unix systems). Access is very low-level and is typically restricted to individual memory read/write operations, low-level thread or process control, and simple trap support. Individual programming models provide support, e.g., the MPI profiling interface and the recently ratified addition of the MPI tool information interface for MPI-3, or CHARM++ with its options to track and record messages between chars. Other programming models, most notably OpenMP, are moving towards standardized interfaces to replace vendor specific solutions, but all those of approaches are programming model specific, provide no support for hybrid models, and require separate tools to be written for each model.

Assessment

Challenges: This work helps address three major OS/R challenges: the support of a wide range of programming and execution models within a single stack; the need to provide adequate tools support to enable efficient machine utilization wrt. a wide range of metrics, incl. speed, power, memory, and reliability; and the need to support hierarchical parallelism through multiple models.

Maturity: The concept of debug interfaces has been around since initial OS designs and the requirements from the tools side are well understood.

Uniqueness: While the general principles apply to any compute environment, its necessity and urgency will be first felt in efforts that experiment with a large number of new and potentially domain specific programming models. The co-design efforts, identified as one of the major approaches to reach exascale, are at the forefront of this research. It is therefore not likely that other areas or communities will produce the required advances in the anticipated timeframe for the problems targeted here.

Novelty: As described above, OS-level or programming model specific approaches exist, but no programming model or even execution model independent approach has been developed or proposed. In particular, the concept of programming model agnostic abstractions and their mapping to model specific constructs is novel and challenging.

Applicability: The results of this project will immediately be useful to applications in other domains or in codes only using a single, conventional model, since it guarantees interoperability of tools. It thereby also makes other exascale research in debugging and performance analysis accessible to a wider audience.

Effort: Investigating this approach is likely to be a multi-year effort and should include research teams from a wide range of communities, including programming models and DSLs (to help define and expose abstractions); runtime and OS layers (to implement the debug interface); and the tools community (to define requirements and develop tools exploiting the new abstractions). The design of the interface must be executed as a co-design effort between these areas, potentially also including hardware and application software architects.